UNIDAD 2:

Teoría de Sistemas

TGS-Enfoque en el Desarrollo de Software

Sistemas en Programación: Conceptos Clave y el Poder del Enfoque Sistémico

En el mundo de la programación, donde cada línea de código y cada módulo contribuyen al funcionamiento de un sistema complejo, resulta esencial adoptar una forma de pensar que vaya más allá de lo inmediato y lo aislado. Aquí es donde la “Teoría General de Sistemas (TGS)” y el “enfoque sistémico” se convierten en herramientas clave para diseñar soluciones tecnológicas que sean robustas, escalables y adaptables.

La TGS no solo nos brinda una forma de analizar sistemas complejos, sino que también nos permite comprender las interrelaciones entre los diferentes componentes que los conforman. Esto es particularmente relevante en el desarrollo de software, donde las aplicaciones modernas no operan en aislamiento, sino que forman parte de un ecosistema más amplio que incluye bases de datos, APIs, redes y usuarios finales.

Además, el “enfoque sistémico” nos impulsa a pensar en soluciones integrales. No se trata solo de cómo un algoritmo resuelve un problema específico, sino también de cómo ese algoritmo interactúa con el resto del sistema para ofrecer una experiencia eficiente y coherente. Este enfoque es especialmente valioso en un entorno donde la demanda por sistemas interconectados y escalables es cada vez mayor.

En esta unidad, exploraremos cómo los principios de la TGS y el enfoque sistémico se aplican al desarrollo de software. Veremos cómo estas herramientas conceptuales nos ayudan a enfrentar la complejidad inherente al diseño de sistemas informáticos modernos y cómo nos preparan para tomar decisiones más informadas y efectivas en proyectos de programación.

Con esta base, no solo entenderás mejor los sistemas que desarrollas, sino que también desarrollarás una mentalidad estratégica para abordar problemas complejos con soluciones inteligentes y bien estructuradas.

Principios Fundamentales de la TGS en el Desarrollo de Software

1.El Software como un Sistema

El software es un sistema abierto que interactúa con su entorno: hardware, usuarios, otros sistemas y datos externos.

Ejemplo: Un sistema de gestión empresarial interactúa con bases de datos, APIs de terceros y usuarios finales.

2.Globalismo o Totalidad

La funcionalidad del sistema depende de la integración y coordinación de todos sus componentes.

Ejemplo: La eficiencia de un sistema CRM (gestión de relaciones con clientes) no depende solo del módulo de gestión de clientes, sino de su interacción con módulos como ventas y marketing.

3.Interdependencia

Los módulos de software están interconectados. Un fallo en uno afecta al sistema completo.

Ejemplo: Un error en la autenticación de usuarios podría bloquear el acceso a toda la plataforma.

4.Homeostasis

Los sistemas de software deben mantener estabilidad frente a cambios internos o externos.

Ejemplo: Implementar un sistema de tolerancia a fallos que permita al sistema seguir funcionando incluso si un servidor falla.

5.Adaptación

El software debe evolucionar para adaptarse a cambios en requisitos, tecnología o entorno.

Ejemplo: Actualizar una aplicación para integrar pagos digitales debido a nuevas preferencias de los usuarios.

6.Equifinalidad

El mismo resultado puede lograrse por diferentes caminos o diseños.

Ejemplo: Un sistema de recomendación puede implementarse usando algoritmos basados en reglas, aprendizaje automático o datos estadísticos.

Aplicaciones Prácticas en el Desarrollo de Software

A. Diseño de Arquitecturas de Software

Aplicar la TGS para diseñar arquitecturas modulares y escalables, como microservicios.

Ejemplo: Cada componente del sistema se desarrolla, despliega y escala de forma independiente.

B. Modelado de Sistemas

Utilizar diagramas de UML (Lenguaje Unificado de Modelado) para representar la estructura y comportamiento del sistema.

Ejemplo: Modelar un sistema de gestión de inventarios con módulos de productos, órdenes y proveedores.

C. Gestión de Proyectos de Software

Usar un enfoque sistémico para identificar interdependencias y mitigar riesgos.

Ejemplo: Un cambio en el módulo de facturación afecta al módulo de reportes financieros.

D. Testing y Calidad del Software

Analizar cómo las pruebas de un módulo impactan al sistema completo

(pruebas integradas).

Ejemplo: Verificar cómo un cambio en el backend afecta la interfaz de usuario.

Beneficios de Aplicar la TGS al Desarrollo de Software

A. Mejor Comprensión de la Complejidad: Permite analizar sistemas complejos desde una perspectiva holística.

B. Mayor Flexibilidad y Escalabilidad: Diseñar sistemas adaptables que puedan evolucionar con las necesidades del negocio.

C. Identificación de Riesgos y Dependencias: Detectar puntos críticos que podrían generar fallos en cascada.

D. Optimización de Recursos: Minimizar redundancias y maximizar la eficiencia mediante la coordinación entre componentes.

ANEXO 1 - Sistema de Comercio Electrónico

Un sistema de comercio electrónico se puede analizar como un sistema basado en los principios de la TGS:

1. Análisis del Sistema

Conceptos de TGS

- Sistema abierto: El programa interactúa con usuarios, bases de datos, sistemas de pago y otros servicios externos.
- Globalismo: El sistema debe diseñarse como un todo integrado para brindar una experiencia de usuario fluida.
- Interdependencia: Los módulos (gestión de productos, carrito de compras, pagos) estarán interconectados.

Componentes del Sistema

- Gestión de Usuarios: Registro, inicio de sesión y perfiles.
- Catálogo de Productos: Visualización y búsqueda de productos.
- Carrito de Compras: Añadir y gestionar productos seleccionados.
- Gestión de Pagos: Procesamiento de pagos con pasarelas externas.
- Gestión de Envíos: Integración con servicios logísticos para calcular costos y plazos.

2. Diseño del Sistema

Conceptos de TGS

- Jerarquía de sistemas: El sistema principal se divide en subsistemas o módulos, cada uno con sus funciones específicas.
- Equifinalidad: Existen múltiples formas de lograr un objetivo; aquí se opta por una arquitectura modular basada en microservicios.

Arquitectura del Sistema

- Frontend: Interfaz web desarrollada en React o Angular.
- Backend: Servidor central en Node.js con Express.
- Base de Datos: PostgreSQL para datos estructurados (usuarios, productos).
- Servicios externos:
  - ✓ Pasarela de pagos (Stripe o PayPal).
  - ✓ API de envíos (DHL, UPS).

Diagrama del Sistema

- Gestión de Usuarios ↔ Backend ↔ Base de Datos

- Catálogo de Productos ↔ Backend ↔ Base de Datos
- Carrito de Compras ↔ Backend ↔ Servicios Externos (Pagos y Envíos)

3. Implementación del Programa Conceptos de TGS:

- Adaptación: El sistema debe permitir la integración de nuevas funciones en el futuro, como un programa de lealtad o nuevos métodos de pago.
- Homeostasis: Implementar mecanismos de tolerancia a fallos para garantizar la estabilidad del sistema.

Fig 1 -TGS - Modelado Conceptual

ANEXO II - Desarrollo de un Sistema de Gestión de Inventarios Aplicando la TGS

En este ejemplo, diseñaremos un sistema de gestión de inventarios para una empresa utilizando los conceptos fundamentales de la Teoría General de Sistemas (TGS). El objetivo es identificar y aplicar los principios de la TGS durante todas las fases del desarrollo del programa.

Contexto

La empresa requiere un software que administre:

- Productos: Registro de artículos, precios, cantidades y categorías.
- Proveedores: Gestión de contactos y pedidos.

- Órdenes de Compra: Registro de transacciones de entradas y salidas de inventarios.
- Reportes: Generación de informes sobre inventarios, ventas y órdenes.

Aplicación de los Principios de la TGS

1.El Software como un Sistema

El sistema de gestión de inventarios es un sistema abierto que interactúa con:

- Usuarios finales (empleados de la empresa).
- Bases de datos para almacenar la información.
- Otros sistemas externos, como plataformas de facturación.

Representación: El sistema se divide en módulos:

- Gestión de Productos.
- Gestión de Proveedores.
- Gestión de Órdenes.
- Generación de Reportes.

2.Globalismo o Totalidad

El sistema debe diseñarse como un todo. Los módulos no operan de forma aislada, sino que dependen unos de otros:

- La gestión de órdenes depende de los datos de productos y proveedores.
- Los reportes dependen de todos los módulos para generar información útil.

Si el módulo de gestión de productos no registra correctamente las cantidades, los reportes y las órdenes mostrarán datos incorrectos.

3.Interdependencia

Cada módulo está interconectado:

- Productos: Los datos alimentan el módulo de órdenes y reportes.
- Proveedores: Facilita la creación de órdenes de compra.
- Órdenes: Afecta las cantidades disponibles en el inventario.

Cuando se registra una orden de compra, el módulo de productos debe actualizar automáticamente la cantidad disponible en el inventario.

4.Homeostasis

El sistema debe ser capaz de mantener la estabilidad operativa:

- Manejar errores (como una conexión perdida con la base de datos).
- Asegurar que los datos estén siempre sincronizados entre los módulos.

Si un usuario intenta realizar una operación inválida (como registrar una orden sin productos), el sistema debe impedirla y mostrar un mensaje claro.

5.Adaptación

El sistema debe ser flexible para adaptarse a cambios futuros:

- Integrar nuevas funcionalidades, como la gestión de códigos QR.
- Ajustarse a regulaciones legales cambiantes (p. ej., impuestos).

El cliente solicita incluir un módulo de seguimiento de envíos. El sistema debe estar diseñado para integrar este cambio sin grandes modificaciones en los módulos existentes.

6.Equifinalidad

El sistema puede diseñarse con diferentes tecnologías y aún así cumplir su propósito:

- Base de datos: SQL o NoSQL.
- Backend: Python (Django) o Node.js.
- Frontend: React, Angular o un sistema basado en escritorio.

Fases del Desarrollo del Programa

Fase 1: Análisis de Requisitos

- Entrada: Entrevistas con los usuarios para identificar sus necesidades.
- Salida: Documento de requisitos que define los módulos del sistema y sus interacciones.

Fase 2: Diseño del Sistema

- Utilizamos diagramas UML para modelar:
  - ✓ Diagrama de casos de uso: Representa cómo los usuarios interactúan con el sistema.
  - ✓ Diagrama de clases: Muestra las entidades principales (Productos, Proveedores, Órdenes) y sus relaciones.
  - ✓ Diagrama de componentes: Define cómo los módulos del sistema se conectan entre sí.

Fase 3: Desarrollo

- Backend: Se utiliza Python con Django para manejar la lógica de negocio y conexiones con la base de datos.
- Frontend: React para la interfaz de usuario.
- Base de Datos: PostgreSQL para almacenar productos, proveedores, órdenes y reportes.

Fase 4: Pruebas

- Pruebas unitarias:Verificar que cada módulo funcione de forma independiente.
- Pruebas de integración: Asegurar que los módulos funcionen correctamente en conjunto.
- Pruebas de estrés: Evaluar cómo responde el sistema a una gran cantidad de datos y usuarios.

Fase 5: Implementación

- Desplegar el sistema en un servidor en la nube (AWS o Azure).
- Configurar accesos para los usuarios y realizar capacitaciones.

Fase 6: Mantenimiento

- Monitorizar el sistema para detectar y corregir errores.
- Adaptar el sistema a nuevos requerimientos.

Fig 2 -TGS - Modelado Conceptual

Bibliografía

Bertalanffy, L. von. (1968). General System Theory: Foundations, Development, Applications. New York: George Braziller.

Skyttner, L. (2005). General Systems Theory: Problems, Perspectives, Practice (2nd ed.). Singapore: World Scientific Publishing.

Boulding, K. E. (1956). General Systems Theory—The Skeleton of Science. Management Science, 2(3), 197–208.

Pressman, R. S. (2020). Software Engineering: A Practitioner's Approach (9th ed.). New York: McGraw-Hill.

Sommerville, I. (2015). Software Engineering (10th ed.). Boston: Pearson.

Weinberg, G. M. (1975). An Introduction to General Systems Thinking. New York: Dorset House Publishing.